The First International Conference on
Machine Learning and Knowledge Discovery (MLKD 2024)
Amirkabir University of Technology, December 18-19, 2024

Transfer Time Reduction for Offloading in the Mobile Edge Computing using Machine Learning

Sorayya Gharravi* Seyed Morteza Babamir†

Abstract

Machine learning can be used to support and optimize operations in edge computing. ML approaches can be used to find patterns in workloads and then use those patterns to improve transfer time, execution time, and response time. Mobile edge computing (MEC) has created a suitable environment for time-sensitive applications and mobile devices. MEC provides services with very low latency compared to the cloud environment. However, due to the time-sensitive nature of the work in the MEC environment and the explosive growth of devices in this environment in the last few years, we still face increased response time and user dissatisfaction. To solve this problem, we need to improve the performance of offloading tasks to edge servers, that is, by finding a suitable edge server to offload tasks and reducing the time of transferring tasks to edge servers.

In this paper, we use machine learning in mobile edge computing to optimize the computational offloading operation to reduce the transfer time and response time. Comparing the proposed method with other proposed methods, the proposed method has a significant improvement in terms of transfer time.

Keywords: Deep Reinforcement Learning, Off Loading, mobile edge computing, Response time, Internet of Things (IoT)

1 Introduction

Machine learning (ML) approaches are suitable for use in various difficult cases and fields, such as resource management in cloud computing and edge computing. Edge computing has rapidly infiltrated various scientific, commercial, social, etc. applications, and has become an integral part of human life. The connection of smart mobile devices to cloud computing in 5G networks has led to the emergence of a new concept called mobile cloud computing (MCC). However, due to the time-sensitive nature of most mobile phone applications and the remoteness of cloud servers to process the requests of these applications and increasing the response time of environment MCC, Given that the main reason for the emergence of the mobile edge computing environment is to enable low-latency processing and acceptable response time for time-sensitive applications at the edge of mobile networks, the criterion of task transfer time to edge servers in this environment is of particular importance. However, with the unprecedented increase in mobile devices in the mobile edge computing environment that want to use edge servers to perform their tasks, and the mobility of these devices, the time to transfer tasks to edge servers and the length of the task queue to use edge server resources for processing increased, which, given the shortage of edge server resources and the time-sensitive nature of most tasks in the MEC environment, increased task execution time and user dissatisfaction. Articles have presented methods for reducing the transfer time in offloading operations on edge servers, which is still relevant with the increasing number of devices in the MEC environment.

In this paper, we propose an optimal DRL-based offloading strategy for computing tasks in a mobile edge computing environment to reduce transmission delays and thus reduce response time. Of course, this optimization is performed in two parts: In the first part, it is first determined whether the tasks are to be processed on the mobile device that generated them, or offloaded to edge servers for processing. In the second part, if the decision is made to offload tasks to edge servers, the most appropriate edge server for processing is selected, taking into account user mobility and the criterion of reducing transmission time.

2 Background Knowledge

In this section, we explain the main concepts presented in the paper.

2.1 Mobile Edge Computing Environment

Mobile edge computing utilizes the network edge (the connection or interface between the device and the local network and the internet) to extend cloud computing services to mobile base stations. Edge servers (devices that provide the entry point to the network) are primarily located at base stations, near the network edge. Edge servers perform the production tasks of mobile devices.

*Department of Software Engineering University of Kashan Kashan, Iran, S.gharravi@grad.kashanu.ac.ir
†Department of Software Engineering University of Kashan Kashan, Iran, babamir@kashanu.ac.ir

Mobile edge computing supports 3G/4G/5G technologies, wired networks, and wireless networks [11]. As shown in Figure 1, mobile edge computing technology uses edge sites to place computing, storage, and network resources close to users. Edge servers consist of one or more physical machines to provide services. Each edge server has a specific range to provide services, and users within this specific distance can connect to this edge server. And receive services with low latency [10].

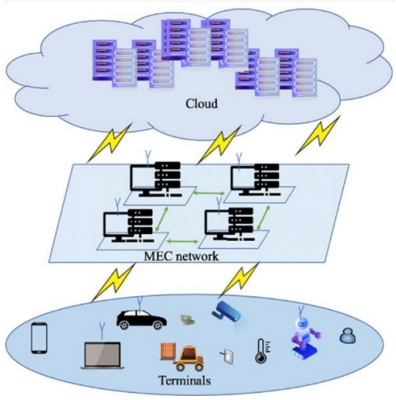

Figure 1: Architecture of the MEC environment

2.2 Mobility in the MEC Environment

Mobility is an essential 5G scenario. The MEC environment also benefits from 5G network services and can provide services to various types of mobile and portable devices. Therefore, different devices that generate dynamic traffic can use these services, connect to edge servers, and move between edges. Figure 2 shows several methods for service operations. Ultimately, our proposed approach supports bot h Case1, Case2, Case 3 and Case 4.

2.3 Task Offloading in Mobile Edge Computing Environment

Offloading computational tasks is a fundamental operation in the mobile edge computing environment, so that applications that require more processing power and storage space than the resources of mobile devices can use the resources of edge servers, thereby reducing processing time and response time [8]. In the mobile edge computing environment, as shown in Figure 3, mobile devices offload computational tasks to edge servers for processing instead of cloud servers, thereby reducing the processing latency and response time. The decision to offload tasks to edge servers needs to consider the latency metric, because mobile device applications are sensitive to latency.If the response delay is high, it is not acceptable to the task requesters, and may not be acceptable because the response was not received within the specified time [6].

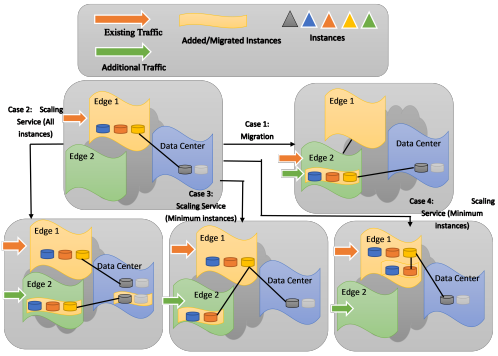

Figure 2: Mobility in the MEC environment

As a result, offloading to edge servers must be reliable in terms of response time latency. If the demand for offloading tasks from mobile devices to edge servers for processing increases, the waiting time, processing time, and consequently the response time will increase, which is not acceptable to users at all. Therefore, optimizing offloading operations to achieve acceptable response time is a requirement of the mobile edge computing environment. Both mobile devices and edge servers (partial offloading) need to cooperate in processing tasks [5]. The mobility of users and mobile devices in the mobile edge computing environment has presented a serious problem for offloading tasks to edge servers, which is: finding the most suitable edge server to offload computational tasks with lower transmission delays. In other words, during the offloading operation of computational tasks, the most suitable edge server should be selected in terms of distance, resources, bandwidth, etc. to reduce transmission delays and response time by considering the mobility of users and mobile devices [9]. We used the DRL algorithm to optimize the computational offloading operation in the proposed method.

2.4 DRL Algorithm

Machine learning (ML) is a program that provides the ability to learn automatically from data and make decisions. RL is a branch of machine learning in which an agent learns how to behave by taking actions, building perceptions, and observing the results in the environment. The combination of the RL algorithm with the deep learning technique has created a stronger algorithm called DRL. The DRL algorithm is one of the methods used to solve loading and scheduling problems at different levels of the cloud environment [9].

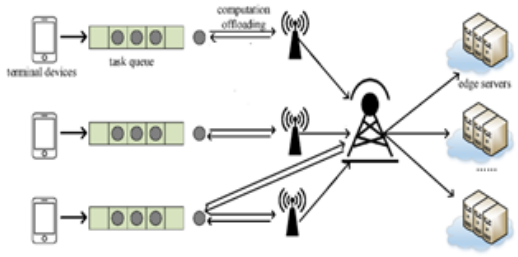

Figure 3: Offloading computational tasks in the mobile edge computing environment

3 Article Contributions

- In the proposed method, the input request is first checked to see if it can be executed on the mobile device. If the result is positive, it will be selected without any transfer time on the mobile device.
- If there is a need to load tasks onto edge servers, the appropriate edge server is selected taking into account the criteria of transfer time, resources, and bandwidth.
- The superiority of the proposed method in terms of transfer time delay over other methods presented in this field is shown in the evaluation section based on different conditions.

4 Previous Work Related to the Research Topic

The articles (Table 1, rows 1-10) have proposed various methods such as machine learning, deep learning, evolutionary algorithms, etc. to reduce the transmission latency of computational unloading operations.

5 MEC Environment Formulation

To implement the MEC environment, we consider the following models.

A. System Model:

The models of mobile devices and edge servers [12] are:

- Set of mobile devices: D=$\{1, 2, \ldots, d\}$
- Set of edge servers: S=$\{1, 2, \ldots, s\}$
- Set of time slots: T=$\{1, 2, \ldots, t\}$
- Duration of each time slot: Δ

Task Model The task model and its parameters [12] are:

- Task$_d$(t): A task from device d at the beginning of time slot t.
- Task$_d$(t)=0: Device d does not enter a new task.
- Size_Task$_d$(t): Number of task bits (task size) Task$_d$(t).

B. Offload decision:

The offload decision model and its parameters are:

- Offload_Task$_d$(t)=0: Process Task$_d$(t) on the mobile device.
- Offload_Task$_d$(t)=1: Process Task$_d$(t) on the edge server.
- Size_Task$_d$(t) (Offload_Task$_d$(t)): Number of bits entered in the mobile device d transmission queue.
- Size_Task$_d$(t)(1 – Offload_Task$_d$(t)): Number of bits entered in the mobile device d computation queue.
- Offload_Task$_{d,s}$(t) = 1: Offload Task$_d$(t) on the edge server s.
- Offload_Task$_{d,s}$(t) = 0: Do not offload Task$_d$(t) on the edge server s.

$$\sum_{d \in D} \left(Offload\_Task_{d,s}(t) = 1(Offload\_Task_d(\mathrm{t}) = 1)\right)$$

Each task can be offloaded to an edge server.

C. Transmission queue:

Transmission queue parameters and its parameters [12] are:

- FIFO: First-in, first-out transmission queue.
- |H$_{d,s}$|2: Channel gain from device d to edge node s.
- PW: Device transmission power.
- W: Bandwidth allocated to a channel
- x^2: Received noise power at the edge server.
- $Rate_{d,s}^{tran} = Wlog_2\left(1 + \frac{|H_{d,s}|^2 PW}{x^2}\right)$: Transfer rate from device d to edge server s (bits per second).

D. Edge Server Model:

The edge server model and its parameters are:

- $Task_{d,s}^{edge}$ (t): Task index in the transmission queue.
- $Size\_Task_{d,s}^{edge}$ (t)) = Size ($Task_{d,s}^{edge}$ (t)) : If the task $Task_{d,s}^{edge}$ (t) be placed in the corresponding queue at time t.
- $Size\_Task_{d,s}^{edge}$ (t): Number of bits entered by the task index in the transmission queue.

Table 1: Computational Offloading in Mobile Edge Computing Environment

Paper	Purpose of the Article	Method Used	Tool Used	Data Set	Evaluation Criteria	Benefits	Open Problem
1 [8] (2021)	Providing a computational offloading method and reducing transmission delay and reducing energy consumption.	Reinforcement Learning	Python programming language.	NavegaporMadrid: http://www.emtmadrid.es	Response Time	Reduce Transmission delay	Improvement of the proposed method with less transmission delay
2 [2] (2021)	Providing an algorithm for offloading computational tasks and reducing transmission delay and access time	Deep Reinforcement Learning	Programming language MATLAB	Hopson One: http://www.thebeijinger.com	Transmission delay, access time	Reduce Transmission delay, Reduce access time	Improving the proposed algorithm with other deep learning methods
3 [6] (2020)	Providing an algorithm for offloading optimal computational tasks and reducing execution time	Deep Reinforcement Learning	Python programming language	Powercastc: http://www.powercastco.com.	Execution time	Reduce execution time	Increasing the accuracy of decision-making in the computational tasks offloading algorithm
4 [14] (2020)	Providing an algorithm for offloading optimal computational tasks and reducing transmission delay	Meta Reinforcement Learning	Programming language MATLAB	https://datasf.org/opendata	Transmission delay	Reduce Transmission delay	Providing algorithms with higher accuracy
5 [7] (2020)	Presenting an online algorithm that optimally adapts offloading decisions to the changing conditions of the Mobile-Edge environment.	Deep Reinforcement Learning	Programming language Python with TensorFlow	Powercastc: http:/www.powercastco.com.	Performance rate, Calculation time	Reduce Performance rate, Reduce Calculation time	Solving resource allocation sub-problems effectively to increase the quality of decision variables
6 [5] (2022)	Providing a resource allocation method in edge computing to reduce time delay and reduce energy consumption	Reinforcement Learning	Programming language MATLAB	Artificial data (random loading)	Time delay, Energy consumption	Reduce time delay, Reduce Energy consumption	Providing a method with less transmission delay and increasing the predictive power
7 [13] (2021)	Providing an algorithm for offloading optimal computational tasks to reduce transmission delay and reduce execution time.	Deep Learning	Programming language Python	https://datasf.org/opendata	Transmission delay, execution time	Reduce Transmission delay, Reduce execution time	Improving the proposed algorithm with other deep learning methods
8 [1] (2020)	Providing an algorithm for offloading optimal computational tasks to reduce transmission delay	Deep Learning	Programming language MATLAB	Hopson One: http://www.thebeijinger.com	Transfer rate	Reduce Transfer rate	Testing the proposed algorithm in dynamic and practical environments
9 [12] (2020)	Presenting an algorithm for offloading optimal computational tasks and presenting a scaling method for resource management to reduce task execution time and reduce energy consumption	Deep Reinforcement Learning	Programming language Python	https://cloud.google.com/free Google Cloud trackers	Task execution time, Energy consumption	Reduce Task execution time, Reduce Energy consumption	Developing more sophisticated methods such as multi-agent deep reinforcement learning
10 [4] (2022)	Presenting a computational offloading method in a multi-user and multi-agent environment to reduce transmission delay	Genetic Algorithm	Programming language MATLAB	https://cloud.google.com/free Google Cloud trackers	Transmission delay	Reduce Transmission delay	Combining the presented method with deep learning techniques

- $Task_{d,s}^{edge}\ (t) = Task_d\ (t)$: Task $\text{Task}_d(\text{t})$ for $\text{t} \in \{1, 2, \ldots, t-1\}$ is unloaded at the edge servers in time interval $\text{t} - 1$. $Task_{d,s}^{edge}\ (t) = 0$: That is, the task is not unloaded.

E. Queues in Edge Servers:

The queue model in edge servers and its parameters [12] are:

- FIFO: Queue model associated with a mobile device in an edge server.
- $Size\_Queue_{d,s}^{edge}\ (t)$: The queue length of mobile device d in edge server s at the end of time slot t.
- $Size\_edge\ _s^{edge}$: The processing capacity of edge server s (in CPU cycles per second).
- $Size\_Queue_{d,s}^{edge}\ (t) = \Big[Size\_Queue_{d,s}^{edge}\ (t-1) + Size\_Task\ (t) - Size\_edge_s^{edge}\ \Delta\ \Big]^+$: Update the transmission queue length.

6 Proposed Method

To speed up the performance of computing tasks in MEC technology, we need to reduce the time of task transfer. To reduce the transfer time, we need to pay attention to two things. First, the computing resources and computing power available to edge servers are limited, and with the diverse computing tasks generated by mobile devices in today's world, and the hardware and computing power of mobile devices are being upgraded, some tasks should be processed by the same mobile device that generated these tasks, if possible, so that there is no transmission delay. However, some large computing tasks must be offloaded to edge servers due to the hardware limitations of mobile devices. There-

fore, the right decision to offload or not offload tasks to edge servers is raised. Second, if the offloading decision is to offload tasks to edge servers, the selection of an appropriate edge server in terms of distance, resources, bandwidth, transmission delays, etc., to offload tasks while supporting user mobility and considering the criterion of reducing the transmission time in the MEC environment is proposed, this operation is based on the parameters defined in Section 5 and shown in Figure 4. Therefore, finding an appropriate computational task offloading strategy to minimize the transmission time by considering the appropriate location for executing tasks (producing mobile device or edge server) and selecting the appropriate edge server to execute tasks in the MEC environment is a challenge. To solve this challenge, in this paper, we propose a method based on the Deep Reinforcement Learning (DRL) algorithm to optimize the computational task offloading operation in the MEC environment, because one of the outstanding features of the DRL technique is the ability to learn automatically and without the need for labeled data, for this reason it can Decision-making without environmental factors of modeling and system dynamics, and is a suitable option for the dynamic MEC environment. In this paper, we consider tasks as indivisible, delay-sensitive, and queuing systems. The proposed method in the MEC environment gradually learns optimal strategies and behaviors by interacting with the environment with unknown load levels, mobility, and complex interactions between tasks, and makes the correct unloading decision. Q-learning algorithms do not work well in environments with many variables. Also, a lot of memory is required to store Q values, which affects the power and speed of convergence. The DRL algorithm has solved this problem. The DRL algorithm is a supervised learning method, which has increased the efficiency and speed of convergence.

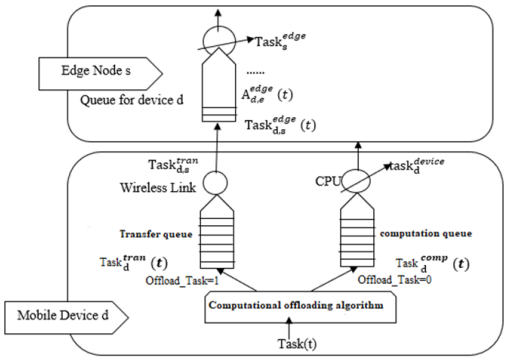

Figure 4: Proposed computational unloading operation

7 Implementation of the proposed method

We used the Linux Ubuntu 24.04 LTS operating system and the IntelliJ community edition environment in Java to implement the proposed method. We used Keras software in Python to implement the DRL algorithm, and Deeplearning4j to convert the neural network code to Java. To collect data, we need to observe and control the MEC environment. For this, we used the Collected tool.

First, mobile devices check their information such as task size, queue length, etc. If a new task arrives, a decision is made whether to run it there, or offload it to edge servers.

7.1 Action

If device d receives a task ($\text{Task}_d(t)$) at time slot t, it must execute one of the following decisions:

(a) The input task is processed by the mobile device itself, or uploaded to an edge server for execution. i.e. $\text{Offload\_Task}_d(t)$

(b) Selecting the appropriate edge server to load and execute i.e. $\text{Edge\_Server}_d(t)$.

Therefore, the decision of mobile device d at time t is shown in equation (1) [12]:

$$\text{action}_d(t) = (\text{Offload\_Task}_d(t), \text{Edge\_Server}_d(t)) \quad (1)$$

7.2 State

We assume that mobile device d can obtain state information including $\text{Size\_Task}_d(t)$, $H\_Task_d^{comp}(t)$ task of device d in the computation queue) and $H_Task_d^{tran}(t)$task of device d in the transmission queue) By controlling external environmental factors [12] in time t through equation (2).

$$H_d(t) = \big(Size\_Task_d(t), H\_Task_d^{comp}(t), H\_Task_d^{tran}(t), Size\_Queue_d^{edge}(t-1)\big) \quad (2)$$

7.3 Transmission Delay

If the parameter $\text{Offload\_Task}_d(t)=1$ for the task $\text{Task}_d(t)$, then Equation (2) shows the amount of waiting time (number of time slots) that should be considered for sending the job. Therefore, the transmission

delay time can be calculated according to Equation (3).

$$H\_Task\ {}_{d}^{tran}(t) = \left[\max_{t\ \in\{0,1,\ldots,t-1\}} H\_TASK_{d}^{tran}(t) - t + 1 \right]^{+}$$
$$Transfer\_Delay = H\_Task_{d}^{tran}(t) * \Delta$$
$$H\_TASK_{d}^{tran}(t) = \min \left\{ t + Rate\ {}_{d}^{tran}(t) + \left\lceil \sum_{s} \frac{Task_{d,s}(t) Edge\_server_{d}(t)}{Rate\ {}_{d,s}^{tran}(t)} \right\rceil - t - 1 \right\} \quad (3)$$

8 Neural Network of the Proposed Method

The goal of the proposed method is to find an optimal path to reduce the transmission time, and consequently the response time. Figure 5 shows the neural network of the proposed method for mobile device d with the parameter vector NETθ_d, which maps from the state H_d(t) to Q-Val to each action. We used three layers to implement the neural network of the proposed method, the input layer, the DRL layer (OffLoading operation) and the output layer. As you can see in Figure 5, the state information is sent to the DRL network through the input layer. In the DRL network, an agent observes the current state and inputs it as input data into the DNN, the DNN outputs the action values. Based on those outputs, the agent selects actions. The parameter vector of the neural network of the device d is represented by the parameter NETθ_d. The details of each layer are as follows.

8.1 Input Layer

The input layer accepts the mobile device state parameters d, which are Size_Task$_d$(t), $Rate\ {}_{d}^{comp}$(t), $Rate\ {}_{d}^{tran}$(t) and $Size\_Queue_{d}^{edge}$(t − 1) as input and sends it to the next layer, and is passed to the DRL layer to determine the appropriate edge server.

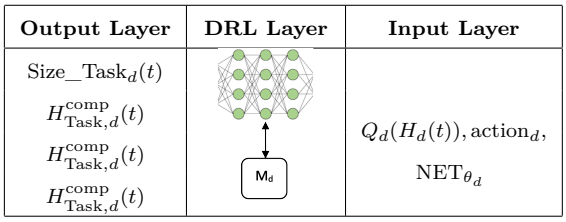

Figure 5: Neural network of mobile device d

8.2 DRL layer

The DRL-based algorithm is also executed on the mobile device d (Algorithm 1) and the edge server s (Algorithm 2). Based on these two algorithms, an attempt is made to select a state that achieves the minimum Q-val, and the task transfer time is minimized.

For device d, a replay memory M$_d$ is considered in the edge server s. The replay memory M$_d$ is the storage location for the history H_d(t+1)) (H_d(t), action$_d$(t) of the mobile device d for the value t.

The edge node s defines two neural networks for the device d ∈ Ds:

a. Evaluation network (NetE$_d$).

b. Target network (Tar_NetE$_d$).

We use the NetE$_d$ network for action selection and the Tar_Q_Val for Tar_NetE$_d$ target network, which approximates the transfer time of an action in a given case. To update the NetE$_d$ evaluation network for future optimal decisions, Tar_Q_Val is used to minimize the difference between Q-Val and Tar_Q_Val. The structure of the NetE$_d$ and Tar_NetE$_d$ neural networks is similar, but their parameter vectors are not the same, the parameter NETθ_d is considered for NetE$_d$ and the parameter NET θ_d^- is considered for Tar_NetE$_d$. Hence, the Q-val parameter for NetE$_d$ is Q_d(H_d(t),action; NETθ_d and the parameter Tar_NetE$_d$ is represented as Q_d(H_d(t),action; NETθ_d). The initialisation of the replay memory Md and the two neural networks is given in steps 2-5 in Algorithm 2.

8.2.1 Algorithm 1 in mobile device d

The parameter V in step 2 of Algorithm 1 represents the number of segments considered. At the beginning of each segment, mobile device d initializes the state according to equation (4), for example:

$$H_d(1) = \big(\text{Size\_Task}_d(1), Rate_{d}^{comp}(1),\ Rate_{d}^{tran}(1),\ Size\_Queue_{d}^{edge}(0)\big) \quad (4)$$

We also define (0)=0 $Size\_Queue_{d,s}^{edge}$ for all edge servers. If device d has a new Task$_d$(t), mobile device d selects its action for task Task$_d$(t) based on equation (5) and the parameter vector NETθ_d from NetE$_d$ [12]. It then sends a Signal request to s_d to update the parameter vector NETθ_d for the task it is using.

$$\text{action}_d(t) = \begin{cases} action\ random\ action\ , & w.p.\ \ \varepsilon \\ argmin\ \ Q_d\ (H_d(t),\ action;\ NET\theta_d), & w.p.\ 1-\varepsilon \end{cases} \quad (5)$$

where with probability ε is a random discovery. Of course, according to the probability $1-\varepsilon$, mobile device D makes a decision that minimizes the Q-val parameter in H_d(t) and NetE$_d$. Mobile device D can see the next state of H_d(t+1) in the time slot (t+1).

Algorithm 1 Offloading algorithm on device d

1: **for** Sec $= 1, 2, \ldots, V$ **do**
2: Give the initial value $Hd(1)$ according to equation (8)
3: **while** $t \in T$ **do**
4: **if** Taskd(t) **then**
5: s_d = req-parameter (send)
6: Receive ← parameter(vector $NET\theta_d$)
7: $NET\theta_d$ = req-parameter (receive)
8: $action_d(t)$ = choose an action according to equation (5)
9: **end if**
10: Consider $Hd(t+1)$
11: Consider $Transfer\_Delay$
12: **for** Task$_d$(t) **do**
13: Send $(H_d(t), action_i(t), H_d(t+1))$ to s_d
14: **end for**
15: **end while**
16: **end for**

8.2.2 Algorithm 2 at Edge Server s

In this algorithm, we first initialize the replay memory M_d and the neural networks NetE$_d$ and Tar_NetE$_d$ related to the mobile device d. Then, the edge server s waits to receive a message from the mobile device d. If the edge node s accepts a request from the device d, it sends NETθ_d to the mobile device d through NetE$_d$. If the mobile device d sends an experience sample (H$_d$(t), action$_d$(t) ,H$_d$(t + 1)) to the edge server S, the server S stores this experience in the replay memory M_d. Note that in this algorithm, we can do the sending of the response parameter (steps 7 and 8) and the training of the network (steps 11 to 19) together. That is, the edge server S sends the vector NETθ_d to when receiving a request, regardless of the ongoing training. The neural network specific to the edge server is trained in steps 11 to 19 by the proposed algorithm. And during training, the NETθ_d vector is updated by the NetEd network based on equation (6) [3]. The edge server randomly samples a set of experiences from the memory (in step 16), which is denoted by p. |p| is based on the experience samples in the set p. The key idea of updating NetE$_d$ is to minimize the difference between Q-Val in NetE$_d$ and Tar_Q-Val, which we denote by $Q_{d,i}^{Target}$, where i represents the step number, are calculated based on the experience samples in Tar-NetE$_d$.

$$P(NET\theta_d\ ,\ Q_d^{Target}) = \frac{1}{|\mathrm{p}|}\sum_{t\ \in\ T}\big(H_d(t), action_i(t); NET\theta_d) - Q_{d,i}^{Target}\big)^2 \quad (6)$$

To derive this Tar-Q-val, let $action_i^{next}$ denote the action Considering the lowest specified Q-value the state H$_d$(t + 1) in NetE$_d$ [3]. As in equation (7):

$$action_i^{Next} = argmin_{action} Q_d\big(H_d(t+1), action; \theta NET_d\big) \quad (7)$$

The Replace Threshold parameter indicates the number of training rounds after which Tar_NetE$_d$ should be updated. That is, for each training round of Replace Threshold, the parameter Tar_NetE$_d$ is updated based on NetE$_d$ (step 18 of Algorithm 2). The goal of continuously updating NETθ_d^-in Tar_NetE$_d$ is to better approximate the transition time parameter in the calculation of Tar_Q_Val in equation (8) [3].

$$Q_d^{Target} = Q_d(H_d(i+1),\ action_i^{Next};\ NET\theta_d^-). \quad (8)$$

Algorithm 2 Offloading algorithm on edge server s

1: Give the initial value to $D_d = 0$
2: Give the initial value to $param\_count = 0$
3: Give the initial value to $NetE_d$ = random θ_d
4: Give the initial value to Tar_NetE_d = random $NET\theta_d^-$
5: **while** True **do**
6: **if** $d\_receive$(req-parameter) = 1 **then**
7: $NET\theta_d$ = receive(device d)
8: **end if**
9: **if** $d\_receive(H_d(t), action_d(t), H_d(t+1)) = 1$ **then**
10: M_d_store_experience($(H_d(t), action_d(t), H_d(t+1))$)
11: **for** each experience in M_d **do**
12: M_d_Gain_experiences($H_d(t), action_i(t), H_d(t+1)$)
13: **end for**
14: $Q_{(d,i)}^{Target}$_Setvector = $Q_{(d,i)}^{Target}$
15: θ_d_Update_minimize = $P(NET\theta_d, Q_d^{Target})$ according to equation (8)
16: $Count++$
17: **if** mod($Count, Replace\_Threshold$) = 0 **then**
18: $NET\theta_d^- = NET\theta_d$
19: **end if**
20: **end if**
21: **end while**

9 Evaluation

In the evaluation section, we evaluate and compare our proposed method, and present experiments based on large-scale simulations with edge-to-edge communications. In this paper, we use the EdgeCloudSim simulator [1] as our simulation environment, and compare the results of our proposed algorithm with those of the papers listed below, which address task loading optimization in the MEC environment. This comparison is based on transmission delay measurements.

- No_OffLoad method

- Random_OffLoad method
- ADRL method [12]
- Genetic method [3]

9.1 Simulation environment configuration

The environment parameter settings are given in Table 2. The neural network settings are as follows. The batch size is set to 16. The learning rate is 0.001 and the discount factor is 0.9. The probability of random exploration is gradually reduced from 1 to 0.01 [12]. For the simulation, we used the EdgeCloudSim optimizer. in these assessments, we consider the environment to be constant.

Table 2: Simulation environment parameter settings

Parameter	Value
D	50 [12]
P_d^{device}	2.5 GHz [12]
S	5 [12]
Δ	0.1 second [12]
P_s^{edge}	41.8 GHz [12]
$\text{Rate}_{(d,s)}^{\text{tran}}$	14 Mbps [12]
$\text{Size\_Task}_{(d)}(t)$	$\{2.0, 2.1, \ldots, 5.0\}$ Mbits [12]
PW_d	0.297 gigacycles per Mbits [12]
$W_{(d)}$	10 time slots (i.e., 1 second) [12]
Task arrival probability	0.3 [12]

9.2 Simulation Results

The results of comparing the proposed method with the methods mentioned in Section 6 with the transmission delay criterion are shown in Figures 6 to 8. In Figure 6, we see that by increasing the task arrival probability parameter from 0.1 to 0.6, the proposed algorithm encounters an increase in the average delay of 21.6%, but the average delay of the evaluated methods increases by at least 38.3%. We see that by increasing the task arrival probability parameter, the increase in the average delay of the proposed algorithm is significantly less than that of the evaluated methods. Figure 7 shows that with the increase in the number of mobile devices, with the increase in the number of mobile devices (increase in the number of tasks produced by mobile devices), the average transmission delay of all the evaluated methods increases. In Figure 7, we see that when the number of mobile devices reaches 130, the average delay is 36% less than the evaluated method. In Figure 8, we reduce the number of edge servers from 5 to 2, and compare the proposed method and the evaluated methods. We observe that the average latency of our proposed algorithm increases by 31.3%, while the other evaluated methods increase by at least 45.5%.

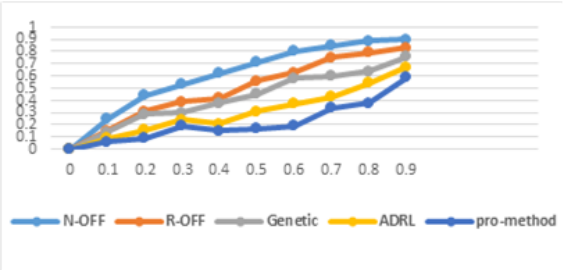

Figure 6: Evaluation of transmission delay under different task arrival probabilities

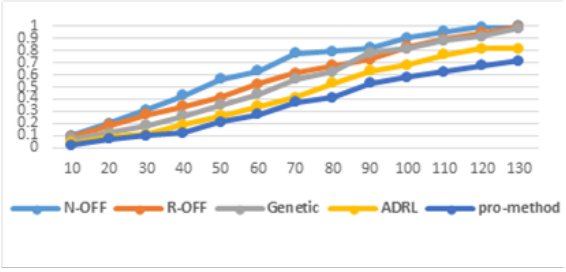

Figure 7: Evaluation of transmission delay under different numbers of mobile devices

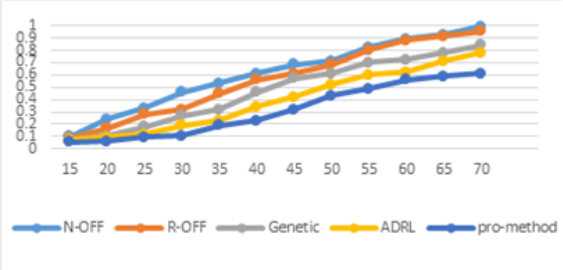

Figure 8: Evaluation of transmission delay under different capacity of edge servers (GHZ)

10 Conclusion

In this paper, we investigated the optimization of computational offloading operations in the MEC environment to reduce the load transfer time in edge servers. This optimization was performed in two stages, the first stage is to determine whether the entered tasks can be executed on mobile devices, if so, they are executed without transfer time on the device itself, otherwise, in the second stage, the proposed algorithm tries to identify the nearest edge server with the minimum transfer time to offload the computational load. We used the DRL algorithm to implement this method. By comparing the proposed method with other methods in the evaluation section, we showed that the proposed method has a significant improvement in the load transfer time in edge servers. We defined the use of another machine

learning method in addition to DRL as future work.

References

[1] N. Alaei and F. Safi-Esfahani. Repro-active: a reactive–proactive scheduling method based on simulation in cloud computing. *The Journal of Supercomputing*, 74(2):801–829, 2018.

[2] Z. Cao, P. Zhou, R. Li, S. Huang, and D. Wu. Multi-agent deep reinforcement learning for joint multichannel access and task offloading of mobile-edge computing in industry 4.0. *IEEE Internet of Things Journal*, 7(7):6201–6213, 2020.

[3] S. Chakraborty and K. Mazumdar. Sustainable task offloading decision using genetic algorithm in sensor mobile edge computing. *Journal of King Saud University-Computer and Information Sciences*, 34(4):1552–1568, 2022.

[4] J. Chen, H. Xing, Z. Xiao, L. Xu, and T. Tao. A drl agent for jointly optimizing computation offloading and resource allocation in mec. *IEEE Internet of Things Journal*, 8(24):17508–17524, 2021.

[5] X. Chu and Z. Leng. Multiuser computing offload algorithm based on mobile edge computing in the internet of things environment. *Wireless Communications and Mobile Computing*, 2022(1):6107893, 2022.

[6] L. Huang, S. Bi, and Y.-J. A. Zhang. Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks. *IEEE Transactions on Mobile Computing*, 19(11):2581–2593, 2019.

[7] L. Huang, X. Feng, C. Zhang, L. Qian, and Y. Wu. Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing. *Digital Communications and Networks*, 5(1):10–17, 2019.

[8] X. Li, L. Huang, H. Wang, S. Bi, and Y.-J. A. Zhang. An integrated optimization-learning framework for online combinatorial computation offloading in mec networks. *IEEE Wireless Communications*, 29(1):170–177, 2022.

[9] W.-x. Liu, J. Cai, Q. C. Chen, and Y. Wang. Drl-r: Deep reinforcement learning approach for intelligent routing in software-defined data-center networks. *Journal of Network and Computer Applications*, 177:102865, 2021.

[10] P. Mach and Z. Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE communications surveys & tutorials*, 19(3):1628–1656, 2017.

[11] T. Soyata. *Enabling Real-Time Mobile Cloud Computing through Emerging Technologies*. IGI Global, 2015.

[12] M. Tang and V. W. Wong. Deep reinforcement learning for task offloading in mobile edge computing systems. *IEEE Transactions on Mobile Computing*, 21(6):1985–1997, 2020.

[13] Z. Wan, X. Dong, and C. Deng. Deep learning with enhanced convergence and its application in mec task offloading. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 361–375. Springer, 2021.

[14] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas. Fast adaptive task offloading in edge computing based on meta reinforcement learning. *IEEE Transactions on Parallel and Distributed Systems*, 32(1):242–253, 2020.